Programming Project 1: Mini-shell

CS415 - Operating Systems

1 Overview

You will implement a simple command-line interpreter with similar behavior to the Linux default **sh** command-line interpreter (otherwise known as a "shell"). Your program must be able to execute commands and redirect the standard input or output of commands from/to files. You do `NOT` need to be concerned with piping the output of commands to other commands or background job control or the append versions of file redirection.

2 Background

Your shell should use the `fork` system call and the `execv` system call (or one of its variants) to execute commands. The shell needs to use either `wait` or `waitpid` system calls to wait for a program to complete execution. You should recognize the command `exit` to mean that your shell program should terminate by calling the `exit()` system call.

A very simple shell such as this needs at least the following components:

- a command-line `parser` to figure out what the user is trying to do.
- If a valid command has been entered, the shell should use `fork` to create a new child process, and the child process should `exec` the command.
- Your shell will need to support file redirection. Use the same syntax as defined in the Bash shell: a single '>' implies that one needs to redirect the standard output of the program being started to the referenced file while a single '<' implies the same with standard input. You do not need to implement support for the Bash pipeline operator '|'.

  Before calling `exec` to begin execution, the child process may have to close the `stdin` (file descriptor 0) and/or `stdout` (file descriptor 0), open file which is being used to read input or write output and use the `dup2` system call to make it the appropriate file descriptor. Don't forget to use the `close` system call to close the old file descriptor.

2.1 More on the command line parser

For functions started from a command-line interpreter, the variable `argc` stores the number of arguments in the command line that launched the program while the array of character strings stores the command and arguments, with `argv[0]` storing the command name, `argv[1]` storing the first parameter, `argv[2]`. storing the second parameter, and so on.

The POSIX (and consequently, Linux) system calls for starting processes have similar function prototypes to `main()`. A command-line interpreter must be able to count the number of arguments in a command line and then convert the command line into an `argv` array.

You will to write a function that accepts a character string as an input parameter and return a pointer to a structure of the following data type:

```
struct arguments
{
   int argc;
   char *argv[];
}
```

This contents of this structure can then be used to construct arguments for one of the `exec()` system calls when you load the new program into the child process.

For performance reasons, shells like `bash` implement some commands directly within the shell rather than paying the penalty of starting a new process. For this assignment, you need only do this for the change directory (change folder) command `cd` and the print working directory (print working folder) command `pwd`.

3 Problem statement

You must write and test the simple command-line shell described in the previous section.

Your program must support the following core features:

- prompt for a command and read a command
- parse the command (setting up argc and argv) and execute the command
- correctly respond to signals such as end of file and break
- implement the *cd* and *pwd* command to change and print the current working directory.

You are required to implement a sufficient test suite to adequately test your solution. You must submit this test suite (and supporting code) as part of your submission. This also includes providing the sufficient makefiles and supporting scripts to build your program from source.

4 Submitting your program

Combine source code, examples of your program executing, and any test scripts and test data into a single PDF document. Attach your submission to the assignment entry in Blackboard. Your submission document must be in PDF format; submission of documents in any other format will result in deduction of points from your grade.